Parallelized Calculation of PI

Christoph Dietzel
Department of Computer Science
Darmstadt University of Applied Sciences
Email: Christoph@cDietzel.de

Christian Happ
Department of Computer Science
Darmstadt University of Applied Sciences
Email: Christian.Happ@stud.h-da.de

***Abstract*—Dummy begins. The parallelized calculation of PI (π) has been chosen to compare different parallel computing technologies by benchmarking the same algorithm running at the same machines. Thus a short program has been written meeting the special requirements of the technologies. Notably these technologies are the MPICH2 implementation of the message passing interface (MPI), the Threads library from the new C++11 standard and OpenMP API from the OpenMP consortium. Dummy ends.**

I. Introduction

The parallelized calculation of PI (π) has been chosen to compare different parallel computing technologies by benchmarking the same algorithm running at the same machines. Thus a short program has been written meeting the special requirements of the technologies. Notably these technologies are the MPICH2 implementation of the message passing interface (MPI), the Threads library from the new C++11 standard and OpenMP API from the OpenMP consortium.

All these standards allow inter process/thread communication which is a crucial requirement to parallelize tasks on modern multi core computer systems. Though all technologies use different approaches.

The MPI is a standardized communication protocol which applies the concept of message passing to communicate between processes. They may be running on the same or on different systems connected by a network. In comparison to the other technologies briefly introduced below, it works on processes and not on threads. The MPI implementation MPIChameleon 2 (MPICH2) released by the Argonne National Laboratory has been used for the benchmarks in this paper.

The latest version of the C++ standard C++11 released in late 2011 supports parallelization by multithreaded programming. The C++11 standard library provides a thread class which takes a function object to start a new thread. The concept for inter thread communication is synchronization. In order to compile the PI calculation program the GNU Compiler Collection (GCC) tool chain has been set up.

The third used technology is the OpenMP API. It makes it possible to parallelize threads by preprocessor directives. The trusted communication concept for the threads is to use shared memory. The same random access memory (RAM) area can be accessed by different threads at the same time.

II. Methods

As benchmarking systems Linux x64 Ubuntu 12.04 boxes with two Xeon E5504 CPUs were chosen. Each CPU has four cores and the hyper-threading technology (HT) allows a maximum of 16 threads per machine. Since MPI provides a interconnection via Ethernet this limitation does not apply for it. The GCC was used in version 4.6.3, the MPICH2 environment in version 1.4.1 and the OpenMP tool chain in version 3.1.

The PI calculation algorithm was taken form Argonne National Laboratory and was adapted to the C++ Threads and the OpenMP environment. PI is the mathematical constant to describe the relation between a circle's circumference and its diameter. If the diameter is 1 then the circumference is π. The keynote for calculation is to approximate with polygons to the circle. Depending on the number of rectangles the calculated result is more accurate.

To achieve meaningful measurement results the number of rectangles was set to $100,000,000$ and $1,000,000,000$ while the number of threads/processes grew by a factor of $p = 2$ $P = \{1, 2, 4, 8, 16, 32, 64\}$. Each setting was measured three times and concluding the average was taken as result. For the MPICH2 benchmark four machines as described above were used. The next section discusses the empirical results of the benchmarks.

III. Empirical Results

The obtained empirical benchmark results for $1,000,000,000$ rectangles with P threads/processes are shown in figure **??**.

The y-axis plots the run time in seconds, the x-axis the number of threads/processes. For one thread/process the run time is the same on any platform. For p where $1 < p < 16$ the measured time is almost equal on all platforms. Just the MPI run time is in average between 10 and 20 milliseconds shorter. For $p >= 16$ threads it is for C++11 and OpenMP slightly longer than with just 8 cores. In contrast the MPICH2 could succeed a further speedup with 16 and 32 processes. Certainly the run time becomes longer with 64 processes. The following table gives an overview about the measured fastest run times, listed independent from the number of

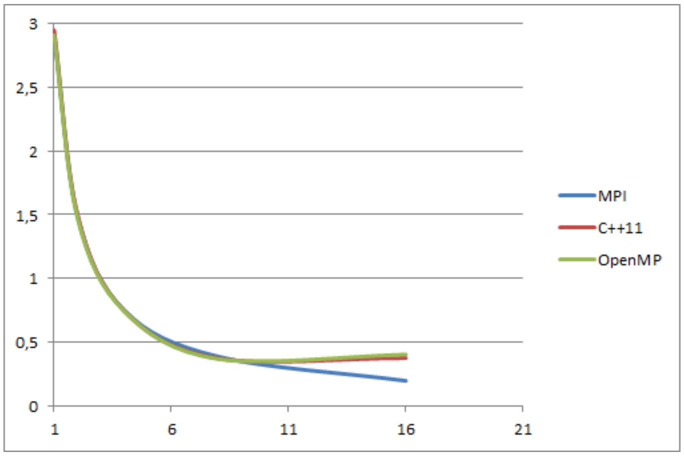

Fig. 1. Benchmark result for 1,000,000,000 rectangles.

threads/processes.

Technology	Fastest Run	Highest Speedup
MPI	0.9263	31.42
C++11	3.6920	7.98
OpenMP	3.6443	7.98

The results for $1 \cdot 10^9$ are almost the same but divided by 10 which follows from the number of rectangles $(1 \cdot 10^8 \div 10)$.

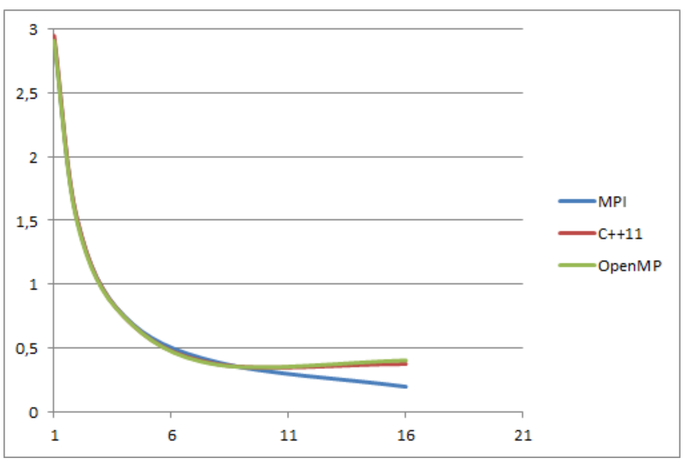

Fig. 2. Speedup for 1,000,000,000 rectangles.

As shown in figure 2 the speedup $S(p)$ is the highest with the MPI technology. C++11 and OpenMP reaching their maximum of a $S(p) = 7.98$ with 8 threads, where MPI's maximum is $S(p) = 31.42$ with 32 processes.

Following the results from the speedup in figure 2 we receive the following efficiency graphs (figure 3). Finally the next section finds a conclusion concerning the introduced values.

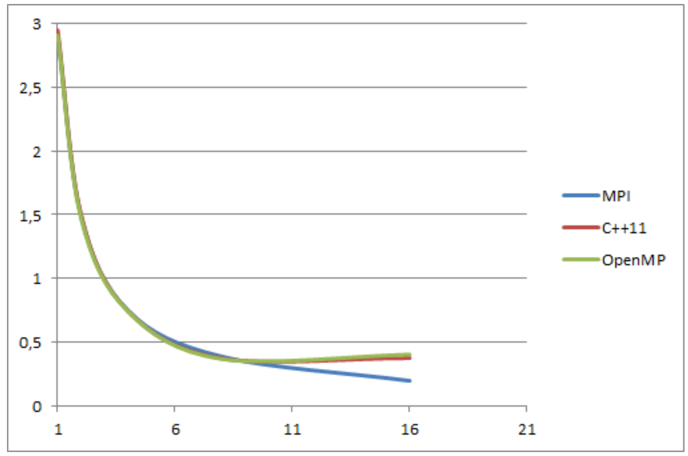

Fig. 3. Efficiency for the values from figure 2.

IV. Conclusion

The conclusion goes here [1].

Acknowledgment

The authors would like to thank Prof. Dr. Moore and Mr. Fugmann for their kind support.

References

[1] A. Mathematics and C. Science. The message passing interface (mpi) standard. online. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/